PCT/GB 2003 / 0 0 4 8 9 2

The Patent Office

INVESTOR IN PEOPLE

68/03/4893

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

RECEIVED
10 FEB 2004
WIPO   PCT

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.
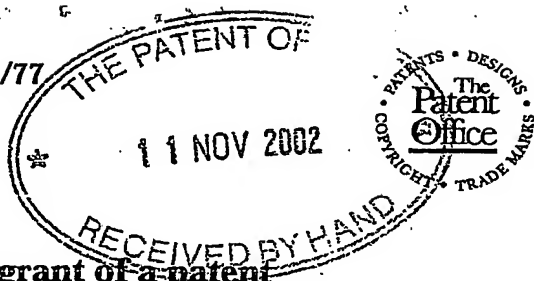
In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed Evens.

Dated   2 December 2003

An Executive Agency of the Department of Trade and Industry

Patents Form 1/77

Patents Act 1977
(Rule 16)

THE PATENT OF
11 NOV 2002
The Patent Office
RECEIVED BY HAND

1/77

12NOV0... 62 67-5 00 47
P01/7700 0.00-0226249.1

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
South Wales
NP10 8QQ

1.	Your reference	HL83718/000/SJR
2.	Patent application number *(The Patent Office will fill in this part)*	**0226249.1**
3.	Full name, address and postcode of the or of each applicant *(underline all surnames)*	CLEARSPEED TECHNOLOGY LTD 3110 Great Western Court Hunts Ground Road Stoke Gifford, Bristol BS34 8HP United Kingdom
	Patents ADP number *(if you know it)*	
	If the applicant is a corporate body, give the country/state of its incorporation	United Kingdom 8484933001
4.	Title of the invention	TRAFFIC HANDLING SYSTEM
5.	Name of your agent *(if you have one)*	Haseltine Lake
	"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	Imperial House 15-19 Kingsway London WC2B 6UD
	Patents ADP number *(if you know it)*	34001

		Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*
6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number			

		Number of earlier application	Date of filing *(day / month / year)*
7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application		

8. Is a statement of inventorship and of right
to grant of a patent required in support of
this request? *(Answer 'Yes' if*
   a) *any applicant named in part 3 is not an inventor, or*
   b) *there is an inventor who is not named as an applicant, or*
   c) *any named applicant is a corporate body.*
See note (d))

Yes

Patents Form 1/77

Patents Form 1/77

9. Enter the number of sheets for any of the
following items you are filing with this form.
Do not count copies of the same document

Continuation sheets of this form   -

Description   40

Claim (s)   -

Abstract   -

Drawing (s)   -

10. If you are also filing any of the following,
state how many against each item.

Priority documents   -

Translations of priority documents   -

Statement of inventorship and right
to grant of a patent *(Patents Form 7/77)*   -

Request for preliminary examination
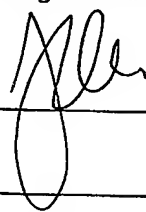and search *(Patents Form 9/77)*   -

Request for substantive examination
(Patents Form 10/77)   -

Any other documents
(please specify)   -

11.   I/We request the grant of a patent on the basis of this application.

Signature   _[signature]_   Date 8/11/02

12. Name and daytime telephone number of
person to contact in the United Kingdom   Mr S Rees   [0117] 910 3200

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 08459 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered 'Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

Patents Form 1/77

Traffic Handling system

Patent summary

Author: Anthony Spencer

Conventions

Convention	Description
commands	This typeface means that the command must be entered exactly as shown in the text and the [Return] or [Enter] key pressed.
Screen displays	This typeface represents information as it appears on the screen and is generally enclosed within a bounding box.
[Key] names	Key names appear in the text written with brackets. For example [Return] or [F7]. If it is necessary to press more than one simultaneously, the key names are linked with a plus (+) sign: Press [Ctrl] + [Alt] + [Del]
Bold-face text	Signal name, Instruction, register or arguments referred to in main body text for purposes of clarification.
	Selections made via the menu hierarchy of a software application.
Words in *Italicized* type	Italics emphasize a point, concept or denote new terms.

Disclaimer

1. Information and data contained in this document, together with the information contained in any and all associated *ClearSpeed* documents including without limitation, data sheets, application notes and the like ('Information') is provided in connection with *ClearSpeed* products and is provided for information only. Quoted figures in the Information, which may be performance, size, cost, power and the like are estimates based upon analysis and simulations of current designs and are liable to change.

2. Such Information does not constitute an offer of, or an invitation by or on behalf of *ClearSpeed*, or any *ClearSpeed* affiliate to supply any product or provide any service to any party having access to this Information. Except as provided in *ClearSpeed* Terms and Conditions of Sale for *ClearSpeed* products, *ClearSpeed* assumes no liability whatsoever.

3. *ClearSpeed* products are not intended for use, whether directly or indirectly, in any medical, life saving and/ or life sustaining systems or applications.

4. The worldwide intellectual property rights in the Information and data contained therein is owned by *ClearSpeed*. No license whether express or implied either by estoppel or otherwise to any intellectual property rights is granted by this document or otherwise. You may not download, copy, adapt or distribute this Information except with the consent in writing of *ClearSpeed*.

5. The system vendor remains solely responsible for any and all design, functionality and terms of sale of any product which incorporates a *ClearSpeed* product including without limitation, product liability, intellectual property infringement, warranty including conformance to specification and or performance.

6. Any condition, warranty or other term which might but for this paragraph have effect between *ClearSpeed* and you or which would otherwise be implied into or incorporated into the Information (including without limitation, the implied terms of satisfactory quality, merchantability or fitness for purpose), whether by statute, common law or otherwise are hereby excluded.

7. *ClearSpeed* reserves the right to make changes to the Information or the data contained therein at any time without notice.

1. Summary of proposals

The document includes proposals for the following inventions:

Programmable 40 Gbits/s Traffic Handler - A traffic handler architecture in which packets are processed by software and inserted into an orderlist for scheduling.

Packet storage system for traffic handilng - A Memory Hub, used for buffering packets in a high line rate Traffic Handler.

State Element - A smart memory cell for serialising accesses to shared state variables.

State Engine - A formal framework for designing an active state storage system using state elements

Programmable orderlist manager - A system for maintaining ordered logical data structures in software at high speeds

Overlapped Virtual Queueing - A low overhead method for setting up and tearing down virtual queues

Notes:

- Figure and reference indexes apply locally within each chapter of this document
- This is a summary document. There will be a lot of additional detail (functions and claims) relating to each proposal which may not be covered in this document.
- In each proposal, additional related design work is also listed. These are ideas which are to be considered to have potential either as sub-patents, or maybe as independent proposals in their own right.

Confidentiality Level: RED

2. A programmable 40 Gbits/s Traffic Handler

2.1 Background

A basic knowledge of the function and anatomy of an internet router is assumed.

A routers switch fabric can deliver packets from multiple ingress ports to one of a number of egress ports. The linecard connected to this egress port must then transmit these packets over some communication medium to the next router in the network. The rate of transmission is normally limited to a standard rate. For instance, an OC-768 link would transmit packets over an optical fibre at a rate of 40 Gbits/s.

With many independent ingress paths delivering packets for transmission at egress, the time averaged rate of delivery cannot exceed 40 Gbits/s for this example. Although over time the input and output rates are equivalent, the short term delivery of traffic by the fabric is bursty in nature with rates often peaking above the 40 Gbits/s threshold. Since the rate of receipt can be greater than the rate of transmission, short term packet queueing required at egress to prevent packet loss. A simple FIFO queue is adequate for this purpose for routers which provide a flat grade of service to all packets.

More complex schemes are required in routers which provide Traffic Management. In a converged internetwork, different end user applications require different grades of service in order to run effectively. Email can be carried on a best effort service where no guarantees are made regarding rate of or delay in delivery. Real-time voice data has a much more demanding requirement for reserved transmission bandwwidth and guaranteed minimum delay in delivery. This cannot be acheived if all traffic is buffered in the same FIFO queue. A queue per so-called Class of Service is required so that traffic routed through higher priority queues can bypass that in lower priority queues. Certain queues may also be assured a guaranteed portion of the available output line bandwidth. The ClearSpeed view of Traffic Handling in context is described in the ClearSpeed Traffic Management system whitepaper [1].

2.2 The problem and prior art

On first sight the traffic handling task appears to be straightforward. Packets are placed in queues according to their required class of service. For every forwarding treatment that a system provides, a queue must be implemented. These queues are then managed by the following mechanisms:

- Queue management assigns buffer space to queues and prevents overflow
- Measures are implemented to cause traffic sources to slow their transmission rates if queues become backlogged
- Scheduling controls the dequeueing process by dividing the available output line bandwidth between the queues.

Different service levels can be provided by weighting the amount of bandwidth and buffer space allocated to different queues, and by prioritised packet dropping in times of congestion. Weighted Fair Queueing (WFQ), Deficit Round Robin (DRR) scheduling, Weighted Random Early Detect (WRED) are just a few of the many algorithms which might be employed to perform these scheduling and congestion avoidance tasks. See reference [2] for a thorough description of these algorithms.

In reality, system realisation is confounded by some difficult implementation issues:

- High line speeds can cause large packet backlogs to rapidly develop during brief congestion events. Large memories of the order 500 MBytes to 1 GBytes are required for 40 Gbits/s line rates.
- The packet arrival rate can be very high due to overspeed in the packet delivery from the switch fabric. This demands high data read and write bandwidth into memory. More importantly, high address bandwidth is also required.
- The processing overhead of some scheduling and congestion avoidance algorithms is high.

Confidentiality Level: RED

- Priority queue ordering for some (FQ) scheduling algorithms is a non-trivial problem at high speeds.
- A considerable volume of state must be maintained in support of scheduling and congestion avoidance algorithms, to which low latency access is required. The volume of state increases with the number of queues implemented.
- As new standards and algorithms emerge, the specification is a moving target. To find a flexible (ideally programmable) solution is therefore a high priority.

In a conventional approach to traffic scheduling, one might typically place packets directly into an appropriate queue on arrival, and then subsequently dequeue packets from those queues into an output stream. The traffic scheduler determines the order of dequeueing. Since the scheduling decision can be processing intensive as the number of input queues increases, queues are often arranged into small groups which are locally scheduled into an intermediate output queue. This output queue is then the input queue to a following scheduling stage. The scheduling problem is thus simplified using a 'divide-and-conquer' approach whereby high performance can be acheived through parallelism between groups of queues in a tree type structure, or so-called hierarchical link sharing scheme [2].

This approach works in hardware up to a point. For the exceptionally large numbers of input queues (of the order 64k) required for per-flow traffic handling, the first stage becomes unmanageably wide to a point that it becomes impractical to implement the required number of schedulers.

Alternatively, in systems which aggregate all traffic into a small number of queues parallelism between hardware schedulers cannot be exploited. It then becomes extremely difficult to implement a single scheduler - even in optimised hardware - that can meet the required performance point.

With other congestion avoidance and queue management tasks to perform in addition to scheduling, it is apparent that a new approach to traffic handling is required.

2.3 Summary of the invention

A traffic handler architecture in which packets are processed by software and inserted into an orderlist for scheduling.
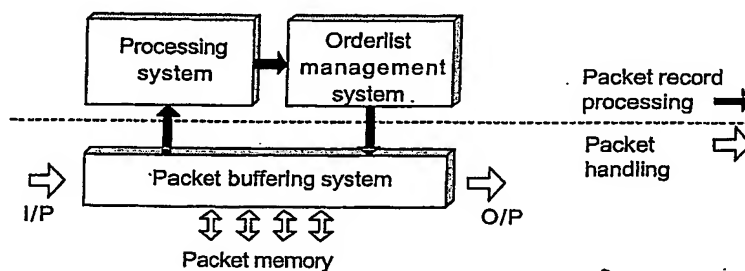
2.4 List of attached figures

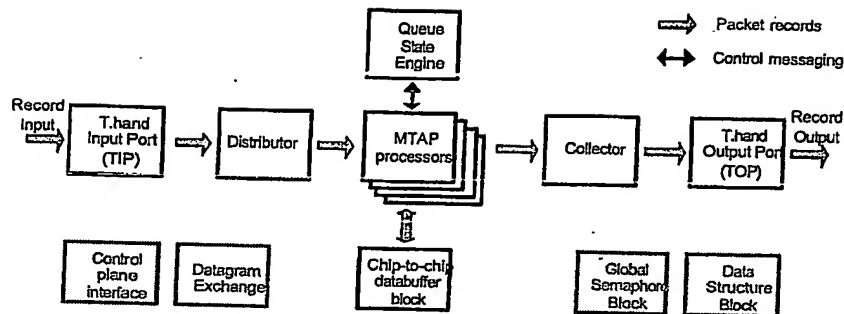Figure 2.1 Traffic Handler system functional overview showing principal components

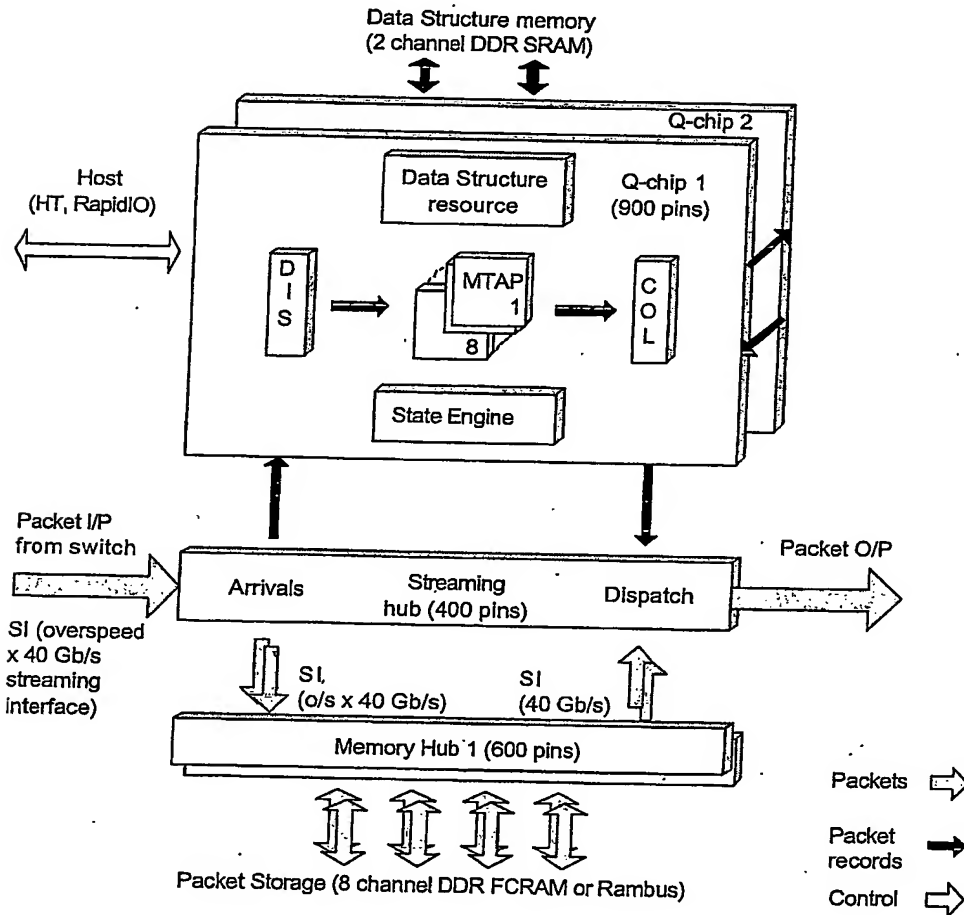Figure 2.2 Functional overview of the system of MTAPs and other ASIC cores in the Q-chip

Figure 2.3 Trafic Handling system implementation

Confidentiality Level: RED

Document Number VRev

© Copyright *ClearSpeed* technology ltd 2001

2.5 Detailed description

Description of the concept and invention

- There are no separate, physical stage1 input queues.

- Packets are effectively sorted directly into the output queue on arrival. A group of input queues thus exist in the sense of being interleaved together within the single output queue.

- These interleaved 'input queues' are represented by state in the queue state engine. This state may track queue occupancy, finish time/number of the last packet in the queue etc. Occupancy can be used to determine whether or not a newly arrived packet should be placed in the output queue, or whether it should be dropped (congestion management). Finish numbers are used to preserve the order of the 'input queues' within the output queue and determine an appropriate position in the output queue for newly arrived packets (scheduling).

- Scheduling and congestion avoidance decisions are thus made "on the fly" prior to enqueueing - a technique referred to within ClearSpeed as "Think first queue later"™.

- This technique is made possible by the deployment of a high performance data flow processor which can perform the required functions at wire speed. The ClearSpeed MTAP processor is ideal for this purpose, providing a large number of processing cycles per packet for packets arriving at rates as high as one every couple of system clock cycles.

Details of the embodiment

Figure 1 shows the MTAP processing system in relation to other components in the wider Traffic handling system.

The packet buffering system and orderlist management system are described in detail in sister patents as each is an innovative solution to a more specific problem.

Figure 2 shows a functional decomposition of the MTAP processing system.

This architecture is described in detail in Application Note 1 of the Per-Flow Traffic Handler design document [3]. This device is referred to as the Q-chip.

Figure 3 shows a full traffic handler implementation using the Q-chip architecture.

Q-Chip 2 is used to implement the orderlist management system. The Memory and Streaming hubs implement the Packet Buffering System.

Additional related design work

There are some additional points to note in our use of MTAP processors to perform Traffic Handling functions. Not sure whether they form claims in this invention, or whether they are patentable ideas in their own right.

Class of service tables: CoS parameters are used in scheduling and congestion avoidance calculations. They are conventionally read by processors as a fixed group of values from a class of service table in a shared memory. This places further demands on system bus and memory access bandwidth. The table size also limits the number of different classes of service which may be stored.

An intrinsic capability of the ClearSpeed MTAP processor is rapid, parrallel local memory access. This can be used to advantage as follows:

- The Class of Service table is mapped into each PEs memory. This means that all passive state does not require lookup from external memory. Enormous internal memory addressing bandwidth of SIMD processor is utilised.

- By performing multiple lookups into local memories in a massively parallel fashion instead of single large lookups from a shared external table there is a huge number of different Class of Service combinations available from a relatively small volume of memory.

- Table sharing between PEs - PEs can perform proxy lookups on behalf of each other. A single CoS table can therefore be split across two PEs thus halving the memory requirement.

In the context of the ordinary use of MTAP processors this is not necessarily innovative. As a tool being used to improve function and performance in a Traffic Handling system this gives a real advantage over the shared memory approach.

Reference material

[1] A. Spencer "Traffic management Whitepaper"

    - Background information on Traffic Management

[2] S. Keshav "An Engineering Approach to Computer Networking", Addison-Wesley, 1997

    - Scheduling, congestion avoidance and hierarchical link sharing theory

[3] A. Spencer "Per-Flow Traffic Handler"

    - Original design work for ClearSpeed Traffic Handling solution

2.6 Key features of the invention

- Traditional packet scheduling involves parallel enqueueing and then serialised scheduling from those queues. For high performance traffic handling we have turned this around. Arriving packets are first processed in parallel and subsequently enqueued in a serial orderlist. This is referred to as "Think First Queue Later"^{TM}

- The deployment of a single pipeline parallel processing architecture (the ClearSpeed MTAP processor) is innovative in a Traffic Handling application. It provides the wire speed processing capability which is essential for the implementation of this concept.

- An alternate form of parallelism (to independent parallel schedulers) is thus exploited in order to solve the processing issues in high speed Traffic Handling.

2.7 Scope of claim

The claim applies most specifically to the use of MTAP processors in a Traffic Handling device used for network traffic management. The claim could be broadened beyond the specific use of MTAPs to cover the more general TF-QL concept and the implementation of the orderlist.

Confidentiality Level: RED

Document Number VRev

3. A packet storage system for traffic handling

3.1 Background

A basic understanding of router anatomy and traffic handling is assumed.

In an egress traffic handler, packets may arrive in bursts at a rate which exceeds the output line rate. Temporary packet buffering is therefore required. By buffering packets in logical queues, different grades of service can be acheived by varying the allocation of the available resources (line bandwidth and memory capacity) amongst the queues.

3.2 The problem and prior art

I am unaware of what prior art exists that might conflict with this proposal. What I do know is that packet queueing and buffering at 40 Gbits/s has been referred to as being "impossible" by other players in the Network Procesing arena. Since ClearSpeed has a solution for this problem then it is reasonable to assume that either a specific idea, or an original combination of ideas described in this chapter must constitute an original solution to a definable problem. The following issues in combination make packet buffering particularly difficult at 40 Gbits/s line rates:

1. High data bandwidth is required to accommodate the simultaneous reading and writing of packets (at worst case fabric overspeed).

2. High address bandwidth is required to cope with the worst case whereby streams of minimum sized packets are simultaneously being written to and read from the memory in a random access mode.

3. Memory capacity must be high as buffers will fill up rapidly during transient bursts at high line rates.

4. The manipulation of state which is associated with either logical queue management or memory management must be minimised at high line rates. The number of system clock cycles typically available to the hardware or software device which performs such a function will be minimal.

A solution which places packets directly into queues mapped into statically assigned memory can meet (2) and (4) but uses memory inefficiently therefore fails on (3). A solution which buffers packets in on-chip memory or SRAM will be able to meet (2) but not (3) since SRAM is a low capacity memory. Implementing a solution which used high capacity DRAM will be able to meet (3) but will have difficulty in meeting (2) as the random access time is small. In attempting to meet (1), solutions need to implement high bandwidth interconnects and high pincount interfaces.

In summary, it is very difficult to design architectures that meet all four criteria.

3.3 Summary of the invention

A Memory Hub, used for buffering packets in a high line rate Traffic Handler.

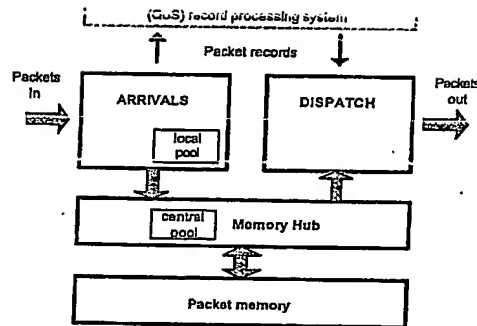**Confidentiality Level: RED** Company Confidential

3.4 List of attached figures

Figure 3.1 Functional overview of the components of the packet storage system

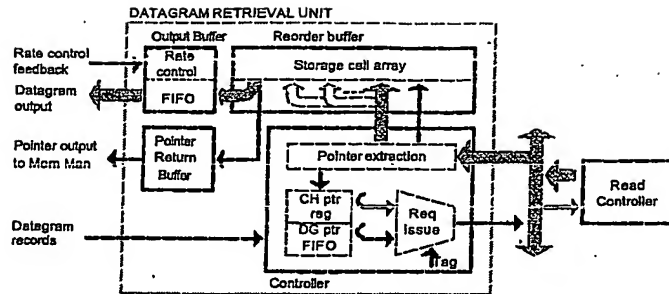*Figure 3.2 Architectural overview of the Memory Hub*

Confidentiality Level: RED

Document Number VRev

© Copyright *ClearSpeed* technology ltd 2001

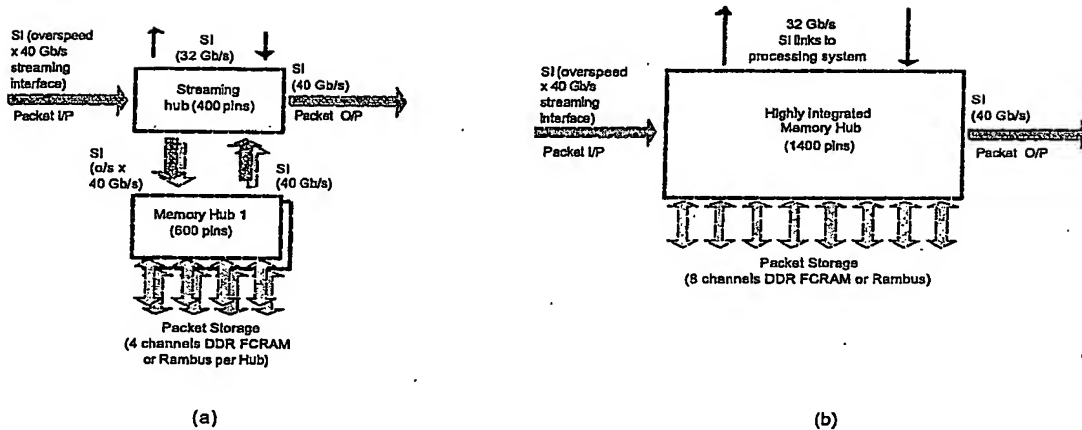Figure 3.3 Datagram Retrieval Unit design

(a)

(b)

Figure 3.4 Implementation of a packet storage system for traffic handling. (a) Multi-chip, scalable implementation approach. (b) Single chip, highly integrated solution.

3.5 Detailed description

This proposal describes an invention in which component behaviours, ideas and devices are assembled into a solution which meets all the required criteria for 40 Gbits/s packet buffering in a traffic handling system. Although certain peripheral behaviours form part of the overall solution, the Memory Hub is the primary embodiment of the invention.

Description of the concept and invention

Memory system decoupling - First, isolate the problem so that it is not entangled and interdependent with other functions.

- Relatively complex functions may be used to control the enqueueing and dequeueing of packets. The complexity of such packet handling/processing can be alleviated somewhat if the packets themselves are not passed around the system. Since access to packet content is not required in traffic handling, the packets can be placed in memory and be represented by small, fixed size packet records. It is these records

---

Confidentiality Level: RED

which are processed and manipulated in logical queues or data structures. The packet records, scheduled in order, can subsequently be used to recover packets for forwarding on the output line.

- The processing and logical queue management functions are thus decoupled from the task of packet buffering and memory management.

- Subsequent QoS processing is performed on a small, fixed sized record of packet metadata. This record typically comprises the packet location in memory (address of first block), the identity of the stream to which the packet belongs (appended to the packet upstream), the packet length and control flags. Additional data is looked up locally using the stream identifier.

<u>Memory management</u> - Memory is generally statically or dynamically assigned when it is used as storage for data structures. Next define an efficient scheme for assigning memory to packets.

- Packet memory divided into small blocks in the memory address space. The blocks may be one of n different configured sizes. For reduced system complexity, n=2 is considered suitable. There is no static assignment of memory to queues. Instead, packets are stored into one or more blocks of a given size (as appropriate). Each block of a given packet points to the next in linked list fashion.

- (It is emphasised here that) packets do not point to one another. In other words, there is no logical queue management.

- The availability of all memory blocks is recorded by a memory manager in the memory hub. To do this the memory manager employs a bitmap - each bit representing a block.

- By reading words from this bitmap, the memory can identify the addresses of free blocks in batches. Bits are converted into addresses, and the addresses held in a central pool of limited but adequate size. This is a form of data decompression which is more storage efficient than maintaining a memory 'freelist'. (Ie. storing all possible addresses in a queue or linked list).

- The central pool can be topped up by either scanning the bitmap, or more directly from the stream of addresses which arrives as packets are read from memory and the memory blocks they occupy are released. If the central pool is full, returned addresses must be buffered and their information inserted into the bitmap.

<u>Efficient packet storage</u> - Placing information into memory will normally require the overhead of updating state which records the presence of the new information. A means of smoothly streaming data into memory at 80G is required which does not get held up by intermediate state manipulation.

- The device that receives packets from the switch fabric is referred to as the Arrivals block. This pipelined processor extracts information from the packet which is used to create the packet record, and slices the packet up into chunks which can be mapped into memory blocks. An appropriate fragment size (and thus memory block size) is selected for each packet based on its length.The packet is forwarded to the Memory Hub and the packet record to the system use for QoS processing and logical queueing.

- Arrivals maintains its own local pool of available memory blocks by periodically reading batches of addresses from the central pool. A separate local (and central) pool is required for each different memory block size implemented.

- This local pool enables Arrivals to load (fragmented) packets immediately into free blocks of memory. The only minor complexity in doing this is to insert the address of the last memory block of the same packet into the current block. It is a simple, fast system which requires no state manipulation other than to pop items from the local pool.

- An important feature of this system is that it supports load balancing across the available memory channels of the Memory Hub. When the local pool is replenished, it receives the addresses of memory blocks which are mapped into different physical memory channels in equal quantities. Consecutive blocks can thus be written to memory blocks in round robin fashion, efficiently spreading the address and data bandwidth load.

- In the unusual event that the local pool becomes empty, packets may need to be partially or fully dropped by Arrivals. These events are still reported through packet record creation so that event handling may be managed centrally by the QoS processor. The processor must purge the packet from memory, and must report the details of any dropped packet. These are both functions the QoS processor already possesses for normal operation.

Confidentiality Level: RED

Document Number VRev

Efficient packet recovery - The packet storage function is deliberately simplified as its 80G performance requirement is challenging. Consequently, the 'open loop' method used by Arrivals makes the 40G packet recovery function more complex. The main issue is that if a packet is stored in multiple blocks in memory, each pointing to the next, one block must be read and the 'next pointer' extracted before a request for the the next block can be issued.

- The device that forwards packets to the line is the Dispatch block. It can be viewed as a type of DMA engine which gets addresses (packet records) from the QoS processing system, uses the address to fetch data (packet) from memory (the memory hub), and forwards the data to the line.

- Memory is nominally organised into (n=)2 sets of blocks - large and small. Block sizes are selected to (a) optimise the usage of memory by matching peaks in the packet size distribution to the block sizes (fairly obvious), and (b) to make retrieval of packets stored as linked lists more efficient (less sol). When a packet is stored in multiple memory blocks there is a delay between the request issued and the first data (which holds the address of the next block) returning. If the block is large enough then the next pointer can be extracted from the first few bytes of a block and the next request issued while the remainder of the block is still being read from memory.

- By selecting a memory block size such that the majority of packets can be stored in a single memory block, efficient aggregate packet recovery can be achieved. Packets can be fetched by a Datagram Retrieval Unit (DRU) in the Memory Hub in a fully pipelined mode (ie. a packet request may be sent before the response to a previous request has returned).

- The recovery of multi-block packets can be made efficient by implementing a hierarchical packet retrieval system which has data recovery occuring at a number of different levels. Dispatch requests complete packets from the Datagram Retrieval Unit (DRU). The DRU fetches memory blocks from the memory read controllers of individual memory channels and reassembles packets. The memory read controllers fetch words from memory in burst read accesses, reassembling the block content by identifying the start and end of valid data within the block.

- For each memory block that the DRU reads, the block address is passed to the memory manager so that the relevant memory map bits can be updated (or the block address reused).

- A Datagram Discard Unit operates in a similar mode to the DRU but does not return data. Its purpose is to update the memory block state bits in the memory manager. This can be done directly for packets stored in single blocks. Packets which are stored in multiple blocks must be read from memory so that the 'next block' pointers can be recovered.

Implementing a viable system - The partitioning of a system must be considerate of real world issues such as device pincounts, power dissipation, silicon process technology, PCB population and routing etc.

- Multiple, independently addressable memory channels are accessed via a Memory Hub. The Hub isolates the memory type and the fabrication technology required for that memory from the rest of the system. The Hub also enables a large, scalable number of memory channels to be implemented as (a) the hub package maximises the number of pins available for memory channel implementation, and (b) multiple hubs may be connected to a single QoS processing chip (see next point).

- The Memory Hub connects to the processing chips (QoS and Queue chips) via narrow high speed serial chip-to-chip links. The burden of memory interfacing on the processing chip pincount is thus minimised, reducing the overall pincount and reducing the packaging cost of a potentially complex ASIC.

Details of the embodiment

Refer to the Per-Flow Traffic Handler design document [1] for further details on the microarchitectural design of a a Memory Hub.

Figure 1 illustrates how Arrivals and Dispatch provide the fork and join points in the packet stream. They isolate the memory hub which simly distributes packet chunks received from the high speed link to memory, or retrieves packets from memory on request.

Figure 2 shows the content and interconnection of the Memory Hub in more detail. The bus acts as a crossbar between the DRUs and the memory channel controllers. This means that packets can be stored in memory blocks on multiple memory channels and fetched/reconstructed by a single DRU. Multiple DRUs exist to increase read

Confidentiality Level: RED

bandwidth and provide load balancing for single line output (OC-768) systems. Alternatively, in multi-line systems (eg. 4 x 10G Ethernet) a single DRU is assigned per line.

The memory manager contains the central pool and the memory bitmap. The memory bitmap would be implemented in embedded SRAM and would typically be 250k bytes in size in order to support a 1G packet memory organised as 512 byte memory blocks.

Figure 3 shows the content and interconnection of the DRU block. The controller supervises read pipelining or the process of packet recovery from a linked list.

Figure 4 shows two possible system implementations. 8 channels of memory provide approximately 20 GBytes/s data bandwidth , and 300 M random accesses per second. This meets requirement for 40G traffic handling (2x switch fabric overspeed). While is is conceivable that the whole system could be implemented in a single device (b), it may be more practical to distribute the function over multiple devices (a). There are many good reasons for doing this, including:

- The physical distrubution and second-level interconect of a large number memory devices on a PCB is alleviated if all devices are not clustered arount a single highly integrated processor.
- It may be difficult to observe tight specifications on electrical characteristics, signal line separation and line termination if memory channels are clustered densely around the perimeter of a single chip.
- Power dissipation is more evenly spread. The combination of high power and pincount in a single device can require expensive packaging technology.

The multi chip approach is also versatile in that the number of hubs used can be scaled to meet the memory requirement of the application (eg. scaling from 10G to 40G, and beyond)

Additional related design work

Reference [1] also describes in detail the Arrivals and Dispatch blocks which could be instanced in a Streaming Hub. Whether these blocks contain intellectual property which should be protected is unclear.

The Streaming Hub: - is an implementation approach which could be used to localise noisy, power hungry high-speed serial interfaces in a single device with minimal additional logic. Dispatch blocks must be co-located on a single chip such as this with connections to all memory hubs so that a crossbar switch between multiple hubs and multiple output lines can be provided.

The Arrivals block: - implements pipelined processing techniques in order to perform rapid, on-the-fly record extraction. A record is forwarded for every packet. Flags in the record are set to indicate whether a packet is fully buffered, partially buffered, or dropped in Arrivals (dependant on the status of the packet buffer). This enables the processor to perform appropriate housekeeping and error reporting.

The Dispatch block: - is a DMA engine which reads a stream of records, retrieves the corresponding packets from the Memory Hub and then forwards them to the output line. Dispatch uses its output buffer occupancy as a servo signal to the memory hub to control the rate at which the Hub delivers packet data.

Reference material
[1] A. Spencer "Per-Flow Traffic Handler"

     - Original design work for ClearSpeed Traffic Handling solution

3.6 Key features of the invention

Document Number VRev

- Queue management and memory management are fully decoupled - (manifested by the packet/packet record concept. This addresses criterion (4))

- There is direct streaming of packets into dynamically allocated memory with almost no first hand state manipulation - (supported by the memory manager and the use of the local pool. This addresses criterion 4)

- Memory address space fragmentation and the availability tracking using the memory managers bitmap provide necessary support for dynamic memory management - (helps to meet (3) through efficient memory usage)

- Usage of high speed serial chip-to-chip links to Memory Hub(s) - (Remote fanout to multiple memory channels enables address and data bandwidth to be scaled to meet requirement without meeting implementation limits - addresses (1) and (2)).

3.7 Scope of the claim

Specifically, a claim is made to a method for implementing a large number of independently addressable memory channels using high speed serial links to memory hubs, to be applied to Traffic Handling on network linecards. Traffic Handling is characterised by high line rates and high speed random access into a large memory.

More broadly, the approach could be applied wherever (latency tolerant) high speed random access is required into a very large memory.

4. The state element

4.1 Background

Situations often arise whereby a function must be performed on a continuous stream of data. If the function is implemented in software on a processor, then each datagram (packet of data) which arrives in sequence from the stream must be stored, processed and then forwarded. This process will take some finite quantity of time to execute. As the rate of packet arrival increases there will come a point at which a single processor can no longer keep up. The function must then either be distributed across multiple processors arranged in a pipeline, or across multiple processors arranged in parallel - each receiving a packet from the stream in turn in some round robin sequence. Packets output from parallel processors are typically reordered before forwarding.

This is fine, as long as there is no interdependence between the processors. They operate independently of one-another, perhaps sharing a common code or data store into which all have read only access.

4.2 The problem and prior art

A problem arises when such processors share state variables for which both read and write access is required. Processors can not be permitted to simultaneously read/modify/writeback a shared variable as the result from the first writeback will be overwritten by the second. It is necessary to serialise the accesses. This raises two significant issues:

1. A system for interlocking processors together must be implemented so that they may arbitrate for a resource and then lock it when there is contention. This control signalling can be complex and add significant functional and performance overhead.

2. When a processor has successfully negotiated for a resource, it should use that resource and then release it as soon as possible to limit the delay imposed on other processors. If access latencies are long to external memories, this can impact heavily on system performance.

Semaphores can be used to interlocck processors, or control logic and caches can be used to intercept concurrent accesses and serialise them; however, these can be complex, slow and/or require significant support tied into hardware. Embedded memory can reduce lock out time, but delays can still be significant.

4.3 Summary of the invention

A smart memory cell for serialising accesses to shared state variables.
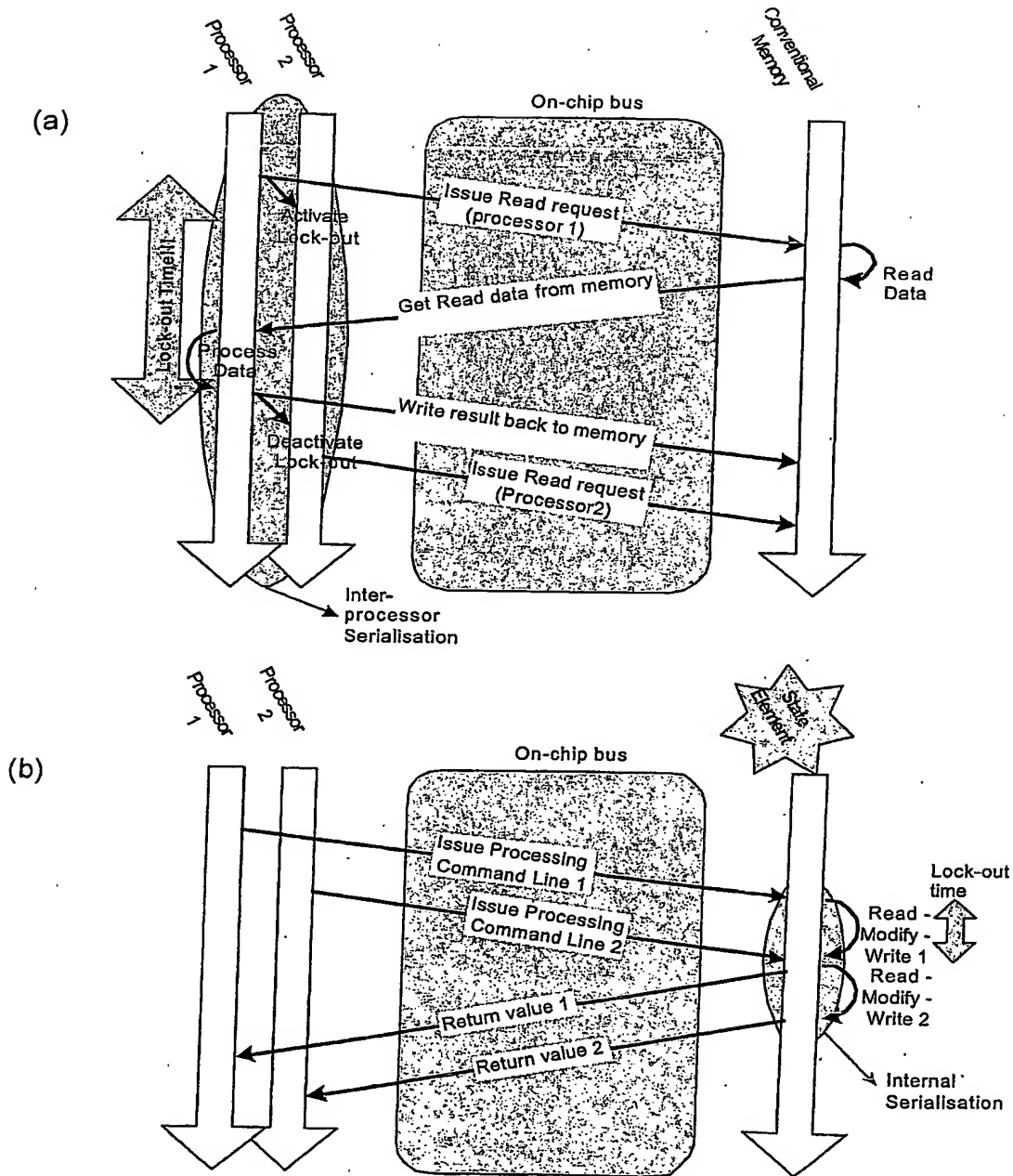
Confidentiality Level: RED

4.4 List of attached figures

Figure 4.1 Illustrating the advantage of the state element concept. (a) conventional approach. (b) using state elements.
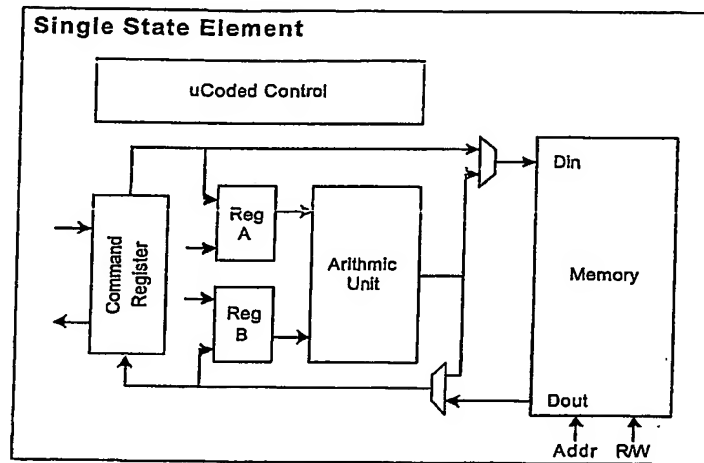
Confidentiality Level: RED

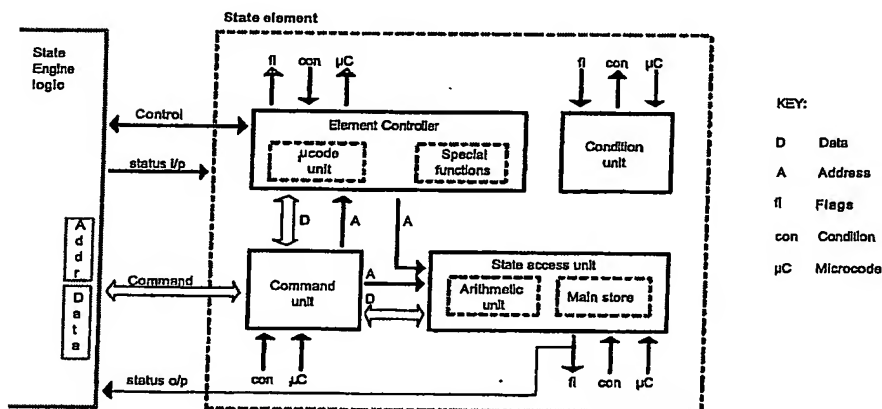Figure 4.2 Functional overview of the state element

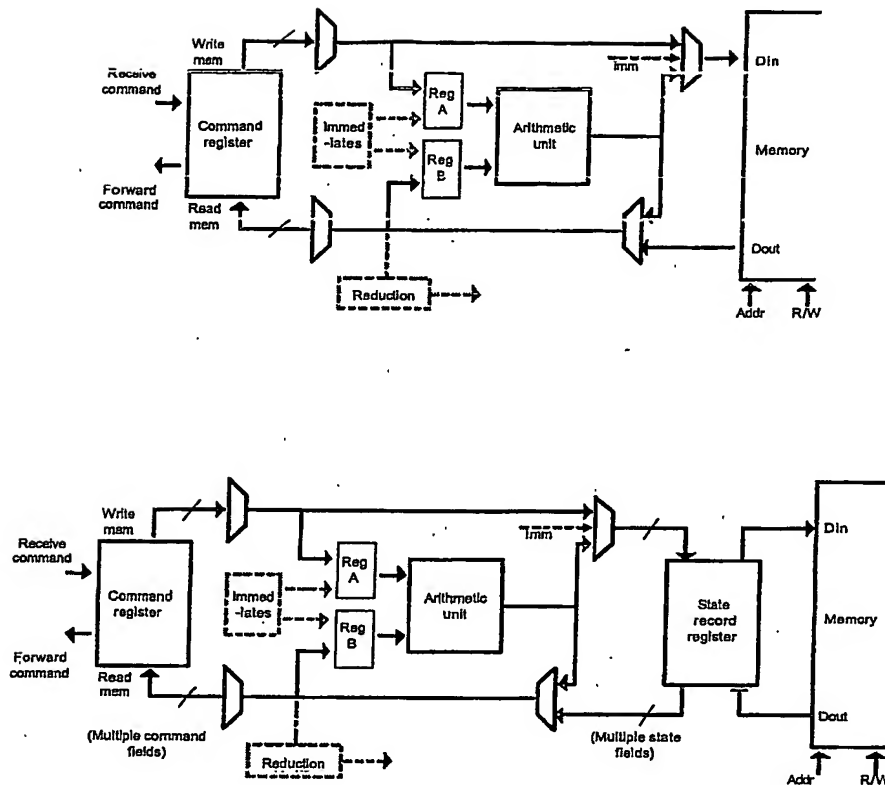Figure 4.3 Implementation overview of the state element

Figure 4.4 Implementation examples of the state and command units

4.5 Detailed description

State elements are the key components which perform the serialisation of acesses into a shared memory. This patent describes only the state element. In a real system state elements must be combined in state engines and connected to the bus. The innovative arrangement of state elements into larger state engines (which can be connected to a system bus) is covered by a sister patent - "The State Engine".

Description of the concept and invention

Instead of getting parallel processors to read from memory, modify, writeback data, get them to request that the memory performs the modification on its behalf. In other words, position the serialisation point not within/between each processor, but in a simple shared processor which has local and rapid access to the memory in which the shared state variables are stored.

The state engine concept and the advantages it brings is illustrated in figure 1.

The state element is analogous to an object in OOD. It has privately stored data which is accessible only via the objects methods. By issuing commands, parallel processors could be considered to be making function calls to the object.

A state element comprises a small block of embedded memory with single cycle read/write access time combined with a simple arithmetic and logic unit. The Arithmetic Unit receives commands (from processors) which comprise an address, data and a command code. The address identifies the state variable which is to be accessed, the data provides operands which simple compute uses to modify the variable, and the command selects a locally stored thread of programmed microcode which is able to read, modify and writeback the state variable within a very small number of system clock cycles. The result can be returned to the processor which issued each command.

These components are shown in figure 2.

Details of the embodiment

Refer to the State Engine design frame work document [1] for full details of the microarchitectural design and implementation of a State Element.

A smart memory element comprises an embedded memory and an attached function - the function could either be hardwired (a finite state machine), or a programmable, microcoded circuit. The latter approach is the more versatile and complex, and receives further attention in this document.

A more complete picture of the system of component modules and their interconnection is shown in figure 3. Note the presence of special function and condition blocks. These greatly extend the functional capability of the element (as described in ref [1]).

The emphasis in state element design is on the rapid memory access speed, not the processing capability. Embedded memory blocks are small enough that single cycle access time is achievable. Configurable R/M/W is possible within a two cycle period as it is possible to perform a simple arithmetic operation on the result of a read and have it turned around for writeback within the second cycle. Typically, a command could be fully processed within typically 3 to 5 clock cycles. Figure 4 illustrates the simplicity of the arithmetic unit, and how the path between the command line and the memory has minimal delay. The lower diagram shows a more complex variant in which multiple items of state are held in memory. The impact on the command line turnaround (and microcode store size) is significant. (However, this is not to say that the lower circuit should not be used. In a lower performance system with a more complex set of state it could be the preferred approach).

Additional related design work

Reference [1] also covers some algorithmic techniques used in conjunction with state elements.

System threads: - Background, system threads could be programmed to operate on the data in the state memory when commands from processors are not being serviced. For instance, could be useful for identifying state entries which are idle.

Find free queue algorithm: - Find_free_queue system function. This is a background thread which implements a "Two strikes and out" algorithm for de-assigning state entries used to represent/manage queues which go idle (ie. empty).

Special function units: - The 'flag unit' and 'address unit' are special function units designed to support the find free queue algorithm. The features they provide are considered to be of generic value and could be used by other algorithms (such as that required for maintaining meters in state elements)

Scheduling algorithm: - The information required by the Self-Clocked Fair Queueing algorithm cannot be mapped directly into the state element. It is represented in a form which makes access and manipulation more robust and efficient. Is this a claim relevant to the state element itself or the software using the state element? (see ref [2]).

Document Number VRev

Reference material

[1] A. Spencer "State Engines - a design framework"

- Original design work for ClearSpeed State Engine solution

[2] A. Spencer "Per-Flow Traffic Handler"

- .Original design work for ClearSpeed Traffic Handling solution

4.6 Key features of the invention

- Intelligent memory - The state element localises the serialisation of parallel data accesses at the memory end, not the processor end. This greatly reduces the latency commonly associated with the blocking of state.

- Functional versatility - The state element provides a number of (configured/)programmed remote functions which may be performed on the stored data - functions would comprise a small number of include data read, write, arithmetic operations and conditional accesses.

- Flexibility - The functions can (but not must) be expressed in microcode so that the state lement remains programmable and does not 'tie' software executing on the processor to functions hardwired into the state lement.

- System efficiency - The read/writeback occurs between the ALU and the memory inside the state element. Only the command travels across the system bus. This reduces the burden on the system bus as compared with conventional approaches.

- System simplicity - The read/modify/write is encapsulated within the state element and serialisation is inherently enforced by the state element logic. Processors can simultaneously issue commands which will cause a function to act on the same item of state without having to first negotiate with one-another.

4.7 Scope of the claim

The problem was identified while using MTAP processors to access shared state in a Traffic Handling application, and state elements were conceived to resolve the contention issue.

It is recognised that contention is not an issue exclusive to Traffic Handling, therefore state elements could be used as a general purpose tool in support of MTAP processors in any application.

Most broadly, contention can arise when any two processors in a realtime (data flow processing) system require R/M/W access to a shared state variable. State Elements could therefore be used in conjunction with any parallel or pipelined arrangement of processors.

5. The state engine

5.1 Background

Situations often arise whereby functions must be performed on a continuous stream of data. If the functions are implemented in software on a processor, then each datagram (packet of data) which arrives in sequence from the stream must be stored, processed and then forwarded. This process will take some finite quantity of time to execute. As the rate of packet arrival increases there will come a point at which a single processor can no longer keep up. The function must then either be distributed across multiple processors arranged in a pipeline, or across multiple processors arranged in parallel - each receiving a packet from the stream in turn in some round robin sequence. Packets output from parallel processors are typically reordered before forwarding.

This is a well proven approach to high performance packet processing, but is limited in its scalability as the number of processors increases. Access to shared memories, be it for code or data, eventually becomes a bottleneck. Simultaneous R/W access to shared state will further add to the complexity of system control signalling in order to resolve contention.

MTAP processors resolve traditional issues relating to instruction lookup, and State Element technology supports parallel processing systems by localising and managing serialisation to shared state. (Both technologies are provided by ClearSpeed Technology Ltd). This leaves the issue of high speed access to multiple items of shared state information by multiple parallel processors. As the number of processors and the complexity of their algorithms increases, address and data bandwidth requirement over the system bus to the shared data will also increase. This can then become a bottleneck.

A good case in point is the challenge of Traffic Management in network routers. A significant, recognised issue in per-flow Traffic Handling is that a number of items of state need to be maintained for each of a large number of queues. The implications of this are that (a) a considerable volume of shared memory needs to be implemented, (b) a lot of memory address bandwidth is required if each queue requires that separate accesses be made to different (shared) state variables, (c) the memory access latency is likely to be long thus causing state blocking during modification to impact on performance.

5.2 The problem and prior art

Contention for shared state variables can be resolved by implementing state elements as described in reference [1] and in the 'State Element' patent proposal. However, the successful implementation of the state element concept in high performance systems requires additional innovation to overcome the following:

1. Arranging processors in parallel can create a high rate of access to the same item of state.
2. What if a given function needs to access to multiple variables from the same address. In other words, needs to access and process a state record?
3. What if multiple functions executing in a processor on a single datagram each require access to different, independently addressable tables of state variables or records?

In short, the fundamental problem being addressed is that of a high rate of state access. This problem must be solved in a flexible way which enables the easy scaling of both the quantity of state being stored and the rate of state access.

5.3 Summary of the invention

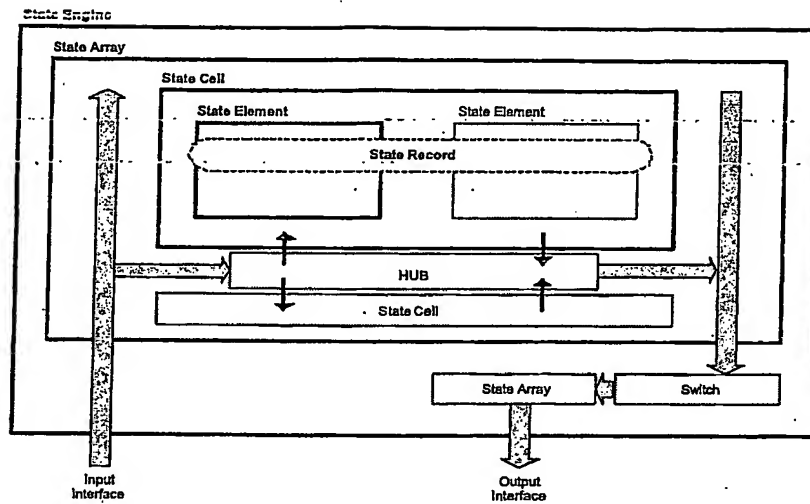A formal framework for designing an active state storage system using state elements

Confidentiality Level: RED

5.4 List of attached figures

Figure 5.1 Conceptual design hierarchy of the state engine

Pipeline state elements in State Cells
for increased processing power

Figure 5.2 Functional view of a state engine

Figure 5.3 Implementation of the state cell
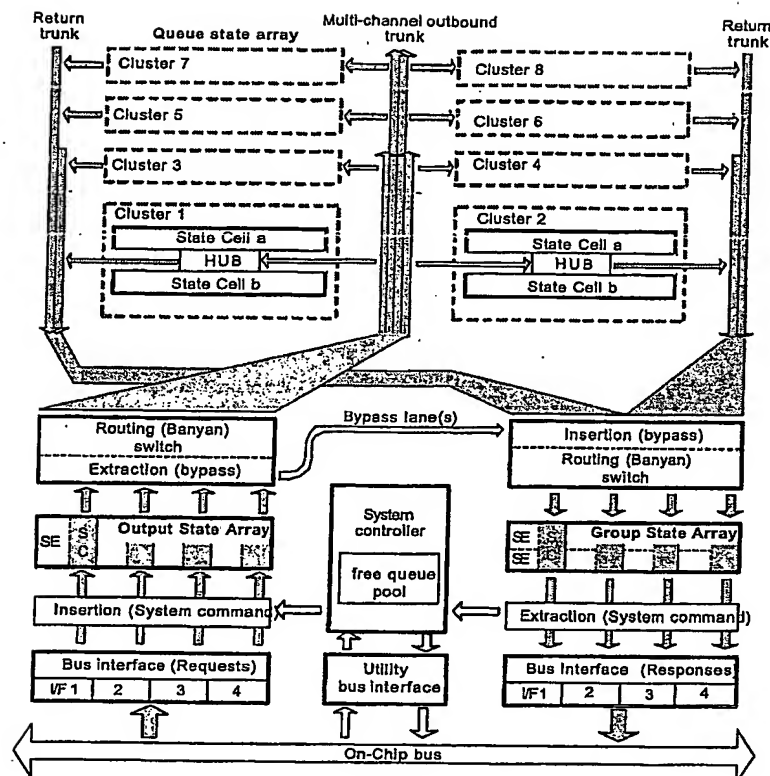
Confidentiality Level: RED

Figure 5.4 The Queue State Engine - an implementation example of a complex state engine design for Traffic Handling and queue management.

5.5 Detailed description

Description of the concept and invention

A state engine can be built up in a structured and well defined manner using the state element as an atomic part. Just as atoms are the components of molecules, which may be the building blocks of simple cells, which then combine into simple organisms - state elements are combined into state cells, which are multiplied into state arrays, which may be grouped together to form state engines.

This hierarchical design framework is illustrated in figure 1. The component parts shown are:

State Record - This is a conceptual entity only. It is a group of one or more state variables which share a common address.

Command line - A message sent by a processor to the state engine. Fields in the command line include command code, address and data. The processor is effectively requesting that the function indexed by the command code be performed on the state record at the given address. Parameters can be both supplied and returned in the general purpose data field.

State Element - As described in reference [1] and the sister "State Element" patent, a state element is a small, private memory which contains state variables accessible only via functions executed by the state elements control logic. Functions typically read a state variable, perform some modification and write a new value back. The

result may also be recorded in a data field in the command line. The primary role of the state element is to manage the state access serialisation point by executing a simple function on memory at maximum speed.

State Cell - If there is more than one state variable in a record, it is permissable for the entire record to be stored as an entry within a single state element. However, as each field in the record would need to be processed in turn this would throttle the available bandwidth to the state. In the State Cell each field of the record is stored as a single state variable in its own State Element. These State Elements are then chained together in a pipeline. The command line passes from one Element to the next, the same address and control word being used at each stage to pick a different field from a common record and perform some function on it. State cell logic provides synchronisation between its constituent Elements which effectively make up a memory oriented pipelined processing system.

The primary role of the State Cell is thus to provide a means of constructing simple, pipelined processors which enable more complex state records to be handled at high speeds.

State Array - The embedded memory used in the State Elements of State Cells must be relatively small in volume for rapid (ideally single cycle) access. This places a limit on the number of instances of a state record which may be stored in a single State Cell. To increase the quantity of state, State Cells of a given type can be tiled to form a large State Array. Scaling during device layout is simplified by the State Array interconnect. The segmentation of an interconnection framework and the coupling of adjacent Cells in a tiled array using well defined interfaces is shown in the accompanying figure. The interconnect preserves order between accesses to the same State Cells. Since order preservation amongst command lines accessing different State Cells is not required, there is no need for the latency of command line accesses to different Cells across the array to be balanced. The Array is scalable in a simple way and is layout-friendly.

Increasing the total state storage volume by multiplying State Cells can also increase overall state access bandwidth as the throughput of an individual State Cell is likely to be a little lower than that of the interconnect. If the number of State Cells is increased to the point that the interconnect becomes the limiting factor then aggregate throughput can be further increased by providing multiple interconnect channels - each channel accessing a different portion of the array (ie. table). This is analogous to designing a memory system with multiple, independently addressable channels to increase random access bandwidth.

The primary role of the state array is to provide scalable capacity. It also provides a means for scaling address and data bandwidth.

State Engine - The State Engine combines State Arrays with all the additional glue logic and facilities which are required to construct a block which can be configured and accessed via a system bus. Components include:

- Bus interface logic
- System control logic - The state engine controller may issue (private) system commands to the state arrays. These commands are invoked by external blocks through accesses to the controller via the utility bus interface. Only (public) state commands may arrive via the main data flow interfaces. System commands configure the arrays or extract diagnostic information.
- Bypass logic - Bypass modes enable commands to skip arrays which they are not required to access. This will conserve power and bandwidth. The required extraction and insertion points can also be used by the system controller.
- Inter array switch connectors - Novel(?) application of (Banyan) switching technology for routing accesses between tables. Only required when there are more than one independent route through each State Array.

State Engine behaviours include:

- Message broadcasting - System commands can be broadcast throughout the memory arrays for retrieving status or passing configuration and control messages. This method is also used for loading microcode into state arrays.
- Multiple accesses - If multiple arrays are connected in a pipe then it is evident that each command line must contain different address and command information for each array. A single command issued from the processor thus results in multiple state accesses.

- Command line "morphing" - As a command lines propagate from array to array they are used and some-times updated as a result of each state element access. The data inserted into the command by state elements in one array could be used by the state elements in the next. Data and perhaps even addresses could be modified.

Details of the embodiment

Example state engine achitectures are documented in the "State Engine design framework" document [1].

The design of the state element is detailed in the S"State ELement" patent.

The design of the State Cell which stitches Elements together in a pipeline is shown in figure 3.

The architecture of the State Array, and the interconnection of State Engine components is illustrated by figure 4. This shows a three-array instance of a Queue State Engine which supports per-flow Traffic Handling.

Additional related design work

Load balancing - It is possible that state records may be allocated dynamically on demand (and also deassigned). If multiple paths exist through a given array then it is desirable for the stored state to be spread evenly across the available State Elements/Cells. The availability of state entries in such a system could be advertised by the Controller in such a way as to ensure that records are assigned from each Element in turn thus balancing the load.

System threads: -

Reference material

[1] A. Spencer "State Engines - a design framework"

- Original design work for ClearSpeed State Engine solution

[2] A. Spencer "Per-Flow Traffic Handler"

- Original design work for ClearSpeed Traffic Handling solution

5.6   Key features of the invention

All of the specified issues associated with high speed data lookup by parallel processors are addressed:

- A formal framework for creating a parallel coprocessor using smart memory (state elements).
- Single access, multiple lookups - A single access acts upon multiple, independent state tables within the state engine. ie. multiple lookups into different tables held in different memories as a result of a single request from the bus.
- Pipelined architecture - Lookups into different tables are not fired off from a point source into different memories. Instead, the access itself (in the form of a command line) is routed from table to table in a serial fashion. It is an object which travels through the QSE.
- Command line "morphing" - As command lines propagate along the pipe from table to table they are used and sometimes updated as a result of each table access. The data inserted into the command by one table could be used by the state elements in the next.
- State cell concept - high throughput pipelined processing (scalable processing power)
- State array concept - 'layout friendly' scheme for scaling quantity of state, bandwidth and load balancing

- State engine concept - multiple orthogonal lookups from a single command uses switching technology for multi-lane state engine architectures.Controller provides system commands for data and instruction broadcasts.

5.7 Scope of the claim

The problems were identified while using MTAP processors to access shared state in a Traffic Handling application. State engines were conceived as a way to arrange the state elements (required for managing state contention) in a way that addressed the additional issue of a high rate of state access.

State Engines can also be architected from the same or similar state elements to meet the needs of other applications - for instance meter management in the related area of Traffic Conditioning. It is therefore speculated that state engines could be used to deliver state element technology to any other application in which parallel (or even pipelined) processors are accessing shared state at high rates.

6. Programmable orderlist manager

6.1 Background

A basic understanding of router anatomy and traffic handling is assumed.

In traffic handling packets may be placed in one of a number of queues. With more than one queue present, a scheduling function must determine the order in which packets are served from the queues. The scheduled order is determined principally by the relative priorities that the scheduler places on the queues - not on the order in which packets arrived at the queues. The scheduling function is thus fairly serial in character.

For example, consider the two popular scheduling methods:

1. Fair Queue scheduling - every packet in the queue is given a finish number which indicates the relative point in time that the packet is entitled to be output. The function that serves packets from the queue must identify the queue whose next packet has the smallest finish number. Ideally, only after the packet has been served and the next packet in the same queue been revealed can the dequeueing function make its next decision.

2. Round Robin scheduling - Queues are inspected in turn in a predetermined sequence. On each visit a pre-scribed quota of data may be served.

6.2 The problem and prior art

The fundamental problem is how to peform such scheduling algorithms at high speeds. A serialised process can only scale with clock/cycle frequency, or by increasing the depth of the processsisng pipe which makes the scheduling decision. This approach to scaling runs out of steam at 40 Gbits/s line rates when the available silicon processing technology may only be able to provide a couple of system clock cycles per packet.

On top of this, the scheduling and queue management task is further confounded by a requirement for a large number of potentially very deep queues. Hardware which executes the scheduling function in a serial manner is then likely to be highly customised and therefore inflexible if it is to meet the required performance.

6.3 Summary of the invention

A system for maintaining ordered logical data structures in software at high speeds

Confidentiality Level: RED   Company Confidential

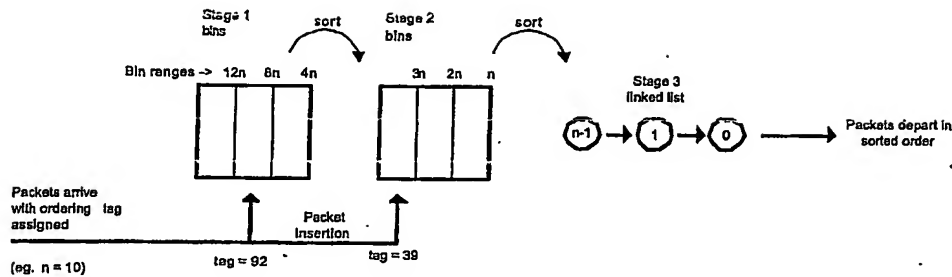## 6.4 List of attached figures

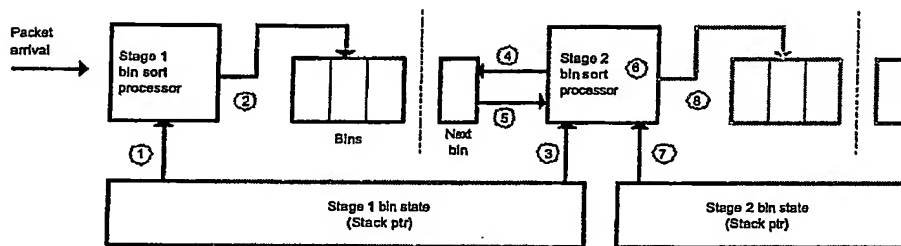*Figure 6.1 Concept of orderlist management using a bin sort approach*

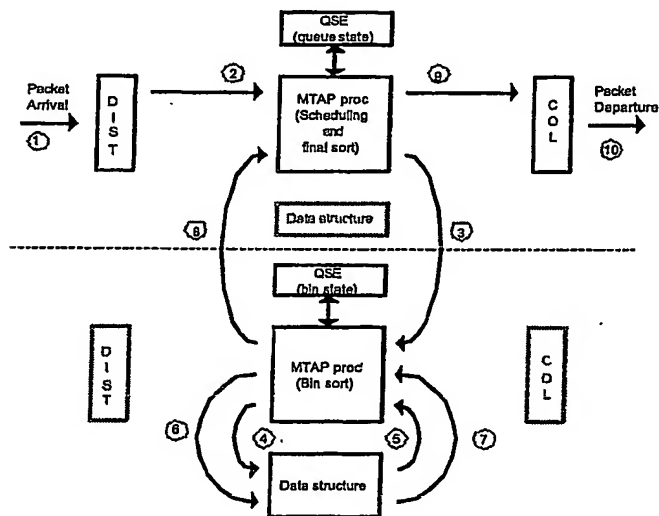*Figure 6.2 Functional overview of an orderlist management system*

Figure 6.3 Implementation of orderlist manager using MTAP processors and state engines

Document Number VRev

6.5 Detailed description

Introduction

A third approach to scheduling is to maintain a single, fully ordered queue instead of multiple FIFO queues. In other words, rather than buffer packets up in a set of parralel input queues and then schedule them in some sequence into and output queue, sort packets on arrival directly into the output queue.

In comparison with scheduling approaches 1 and 2 above, this would appear to be much more difficult. Not only must calculations be made at wire speed for each packet prior to enqueueing, packets must be inserted into a potentially huge ordered data structure.

However, this approach enables parallelism to be exploited in the implementation of the solution. When performance can not longer be improved through brute force in a serialised solution, the way forward is to find an approach which can scale up through its parallelism.

The "programmable 40 Gbits/s Traffic Handler" proposal describes an innovative parallel processing architecture which can provide a sufficient number of processing cycles per packet to enable the finish number calculation to be made at wire speed for packets as they arrive from the switch fabric. This proposal describes a solution to the other half of the problem – the maintenance of a large orderlist at high speed.

Description of the concept and invention

Figure 1 shows the basic concept of bin sorting:

- Consider a small set of bins. Each bin is used to contain packets with a certain range of finish numbers. The content of a bin is not ordered.
- A function is required which receives packets and places them in the appropriate bin.
- Another function is requred which reads the content of each bin in turn in ascending order of the finish number ranges.
- Assume that just as bins are emptied at one end of this sequence, new bins are installed at the other as packets arrive with finish numbers which are, on average, constantly increasing in value.
- Thus, a stream of packets are arriving with randomly varying finish numbers. These are sorted into bins. A stream of packets is output in a coarsely sorted order which depends on bin size.
- The final stage bins can relatively easily be sorted into actual order for output.

Figure 2 shows an approach that applies this concept. The numbering shows the sequence of events as packets arrive, state is accessed, and packets are binned etc. (Full walkthrough could be provided if necesary)

- Each bin could be implemented as a FIFO queue or LIFO stack in memory. Such data structures may be managed by pointers which locate the insertion and removal points for data to/from the structure.
- The functions that operate on the bins need access to these pointers. The functions could be mapped into processors and the pointers into a state memory.
- A data structure is proposed which comprises more than one set of bins. Within a set of bins the finish number range is constant, but between sets the ranges get progressively smaller. Bins with the widest range have the largest finish number values and bins with the smallest range have the smallest finish number values. For example, the total range of finish numbers across all bins in one set may equate to the range of a single bin in an adjacent set.
- When a bin is emptied, it is sorted into the next set of bins.
- Either this is repeated until the finish number range of the final set of bins is unity, OR when the smallest bins are empties they are subject to a final sort before forwarding in order.

Confidentiality Level: RED   Company Confidential

Details of the embodiment

Figure 3 shows how MTAP processors can be used to implement an orderlist manager. The numbering shows the sequence of events which occur as packets are scheduled, binned, re-binned, sorted and output etc. (Full walk-through could be provided if necesary)

- When MTAP processors are arranged in a data flow processing architecture they are well suited to the processing of a high speed stream of packets. They naturally operate by performing batch reads of data [1], doing some processing, and then pushing data out onto queues.

- State Engines used as hardware accelerators can enable the MTAP processors to store and manage the logical state required for the bins.

- The bins are most conveniently implemented as LIFO stacks. This minimises the required state per bin, and simplifies the management of bins as linked lists in memory.

- When each packet is stored in a bin its location in memory is retained in the state engine. This can be used as a pointer by the next packet which needs to be written to the same bin. Each bin is thus a stack in which each entry points to the next one down.

- A databuffer block is used to store the bins. The block contains a bin memory and presents producer and consumer interfaces to the processor [1]. The consumer receives a stream of packets and simply writes them to a supplied address. The producer receives batch read requests from the procesor and outputs data from the requested bin.

- As each bin is organised as a linked list, it is the responsibility of the producer to extract the linked list pointer from each packet as it is read from the bin. Using SRAM the access time should be fast enough to make this serialised process efficient.

- In a real system embodiment it is not necessary to store the actal packets in the bins. Small records which represent records can be processed in their place. This is described in [2]. This simplifies implementation as the bins now store small entities (records) of fixed size.

Additional related design work

On-demand load balancing: The MTAP processors are split between the enqueueing (scheduling) task and the dequeueing (final sort) task. A sufficient number of processors must be implemented in order that they can cope with the transient worst case rate of packet arrival. However, the nominal arrival rate is much lower. This would mean that a number of processors could routinely lay idle or be underused. The proposal is that a small number of processors are assigned permanently to either the enqueueing or dequeueing tasks. The remainder may float. If input congestion is detected then the floating processors thread switch and assist in the enqueueing task. When the congestion is cleared, the floating processors migrate to the dequeueing task and help to clear the backlog in the queues. If dequeuing is well resourced, then floating processors may default to peripheral tasks such as statistics pre-processing for subsequent reporting to the control plane.

Shadowed memory management: This is an essential element of the orderlist management system solution. Although simple, I felt it might be sufficiently valuable as an idea to descibe it separately. Any given data structure needs functions to read and write entries, logical state to characterise the structure, and underlying memory management to efficiently store the structure. The MTAP processor and accelerator only acheive the first two of these. No mention has yet been made of maintaining a freelist of available memory and allocating memory for the data structure to grow into. This in itself can often incur considerable overhead. The efficiency of the orderlist manager is only possible because the memory management has already been performed for it as follows:

Background:

- In 40G traffic handling is is practical to divorce the packet buffering from the processing task. As described in the 40G programmable Traffic Handler proposal, packets are stored in memory within the packet buffering system. Small records are passed to the processing system which efficiently manipulates records in place of packets they represent.

- The packet memory is partitioned into small blocks of fixed size. A free list or bitmap is maintained which keeps track of which blocks are allocated and which are free.

- The bitmap is used by the memory management system to dynamically manage memory. Packets can ber streamed directly into memory on arrival from the switch fabric, with the small record of metadata retained for processing.

- Most significantly, the record will contain the memory address of the (first) block of memory in which the packet is stored.

Packet record handling and storage

- Packet records are stored in a data structure by the orderlist manager. This will require the existance of two resources - storage for the logical state describing each bin in the structure, and storage for the records themselves.

- State storage and bin manipulation are implemented by the QSE and MTAPs respectively. Bin memory management relies directly on the packet memory management.

- Bin memory management concept

- The memory provided for record storage is organise d so that it mirrors the memory provided for packets. For each memory block in the packet memory there is a corresponding location in the record store at a directly related memory address.

- When a packet is stored then the memory·block into which it is placed must be free. The location in the record memory must also then be free.

- When the record is scheduled, the memory system recovers the packet and the memory block occupied by that packet is released. Simultaneously, the corresponding record location is released. Because the storage and retrieval of the packet record is effectively "nested" within the time over which the packet is stored and recovered, the system is very robust.

Use of pointers:

- As records are randomly stored within the record memory, the records belonging to a given bin must point to one another in a linked list arrangement.

- The record contains the pointer to the packet memory. The pointer also then points to the records own location in its memory.

- In effect, a record both points to itself and also to its neighbour. The same information is being stored twice. Consider records A and B which are adjacent in a linked list. Record A has pointer 'Self_A' which is its own location, and 'Next_A' which is a pointer to the next record in the list (record B). Record B also has 'Self_B' and 'Next_B'. It can be seen that 'Next_A' is the same as 'Self_B'.

- Only the 'Next' pointer is actually required in each packet. When a bin is read (in order A, B, C....) each record can have its own pointer identity restored by retrieving it from the record before it in the list. This provides a considerable reduction in the record storage requirement.

Key points:

- Two storage systems can share the same memory manager when the write/read accesses to one are nested within the write/read .accesses to the other.

- Implied data - A records own pointer identity when it is not stored is interchangeable with a linked list pointer when the record is stored. A translation must occur when the record is passed in and out of storage.

Software algorithms for bin management - Novel algorithms for managing the bins have been/are being developed by Ken Cameron. This work might either form part of this proposal, or a proposal in its own right. Refer to reference [3] for further details.

Reference material

[1] ClearSpeed "Network Processor architecture document - V1.0"

    - Architecture proposal for ClearSpeed 40G Network Processor

[2] A. Spencer "Traffic Handler architecture document - V0.1"

    - Architecture proposal for ClearSpeed 40G Traffic Handler (in progress)

[3] K. Cameron "Software Queueing for Traffic Management"

    -Discussion document

6.6 Key features of the invention

- A single orderlist is used instead of multiple queues.
- A method of iterative binning is described which makes the management of large orderlists very efficient.
- Orderlist management is performed entirely in software using MTAP processors and accompanying state engines.
- The processing and state resource can be partitioned to provide either single or multiple orderlists.

6.7 Scope of the claim

The claim is considered to be original on two fronts:

Firstly, it is an alternate way of queueing data. Packets (or packet records) are queued directly into an orderlist instead of in separate per-stream queues.

Secondly, it is unlikely that MTAP processors have been used previously to perform the queue management - either in terms of the process of making the binning decisions, or in the use of state engines for bin pointer management.

Because the invention relies on binning and data structure management in software, it is also speculated that alternative data structures could be mapped into the hardware resources and managed by different software processes. This implies that the invention could have broader application beyond that of suppoorting Traffic Handling.

7. Overlapped virtual queueing

7.1 Background

Some prior understanding of Traffic Management, Traffic Handling and (virtual) queueing would be a benefit.

Per-flow traffic handling requires the independent queueing of traffic belonging to hundreds of thousands, if not millions of different connections. Virtual queueing is a text book approach in which a limited number of physical queues are shared dynamically between a much larger number of connections. In virtual queueing a queue is assigned from a common pool to a stream (flow or flow aggregate) when that flow becomes active. Conversely, if an assigned queue remains empty for a given duration, it is effectively inactive – an unused resource which should be returned to the pool. Observing that a finite memory volume limits the number of packets which may be buffered at any given time, it then follows that a finite pool of shared physical queues can be used to support a much larger number of end-to-end connections as the connections can not all simultaneously have traffic backlogged in the traffic handling system.

So, the idea is that if a connection does not actually have traffic backlogged in a queue at a given point in time, then its queue is unused and might just as well not exist. Queues are allocated and deallocated on a per-demand basis. A queue is only assigned to a connection when that connection has backlogged traffic.

7.2 The problem and prior art

The implementation of virtual queueing presents its own problems:

1. How are queues deassigned? This could be tricky if there are a number of points in the handler at which packets belonging to a given connection could be buffered.
2. How are queues assigned to new connections? Packets belonging to new connections could appear and make an on the spot request for a queue.
3. The purge that is necessary in-between a queue being deassigned and it being assigned to a new connection requires significant system wide messaging and state synchronisation.

This last issue is the core focus of the ClearSpeed "Overlapped Virtual Queueing" concept.

Consider the simple high level view of traffic handling behaviour illustrated in figure 1.

Stream labels attached to packets arriving at A are used to look up a destination queue identifier in B. This queue identifier is then used at C to access a table of queue state in D thus enabling the packet to be appropriately enqueued in E. Subsequent de-queueing by F must also access the queue state associated with each packet when it is served.

There is a close relationship between the information held in B and the organisation of state in D. In the context of virtual queueing these relationships are more specifically:

- Lookup entries in B must be created when packets with unrecognised stream labels arrive at A. These entries must point to an available Q-state entry in D.
- Based on accesses from both C and F, D must be capable of determining (a) whether a queue is empty, and (b) whether the queue is eligible to be de-assigned and returned to the pool in B.
- When D de-assigns a queue, then the related entry in B must be removed.

The implementation of these behaviours should address the problem of pipelining effects caused by packet/message buffering within all links shown in the figure – ie. packets of a newly assigned/deassigned queue can persist

in buffers between A, C and F. The virtual queueing solution must also impose minimal overhead on the system in terms of logical complexity, messaging bandwidth and the storage of additional state.

7.3 Summary of the invention

A low overhead method for setting up and tearing down virtual queues
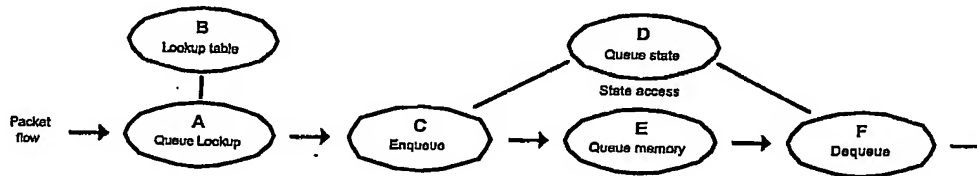
7.4 List of attached figures

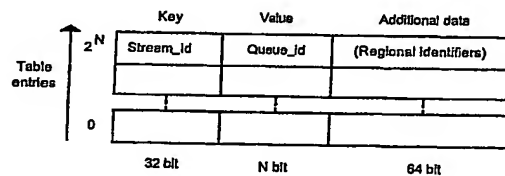*Figure 7.1 Simple schematic view of traffic management*

Figure 7.2 Queue lookup table

7.5 Detailed description

Introduction

Conventionally, one might deassign a virtual queue from an inactive connection, wait for any residual packets for that connection to be purged from the queues, and then reassign the queue to a new connection. This is possible but can require a lot of control signalling, additional state and synchronisation across the system. Overlapped virtual queueing eliminates the purge purge phase and make deassignment and reassignmnet simultaneous.

Instead of a queue being either assigned or unassigned to a connection, it is either assigned or pending re-assignment. Only after boot-up might a queue actually be unassigned. This means that a queue normally always belongs to a connection.

In the pending reassignment state it may still be used by the old connection (that is, assuming the previously inactive connection suddenly comes back to life!).

At the moment of reassignment a new connection takes ownership of the queue and the old connection may no longer place any further packets in it. The old connection will be granted a new queue as and when further packets arrive.

Deassignmnet is thus implicit in reassignment - there is no explicit messaging involved.

Confidentiality Level: RED
Document Number VRev
© Copyright *ClearSpeed* technology ltd 2001

Instead of purging the residual packets of the old connection from the many places that they could be hidden in the traffic handling pipeline, those packets are left alone and are effectively inserted at the start of the new queue. The handover is managed in a controlled fashion to minimise the disruption to the new connection, with old packets perhaps being prevented by software from causing certain disruptive state modifications when they are dequeued.

Description of the concept and invention

<u>Queue lookup</u> :

As queues are dynamically assigned, packets must be tagged with an identifier which identifies them with a specific traffic stream, connection or class of service. This tag enables the packet to be mapped to an assigned queue through the use of a lookup table. A table with behaviours which specifically support overlapped virtual queueing is illustrated in figure 2 and described as follows.

The table can be indexed in one of two ways.

1. An entry is identified by content addressing using the *key*.

2. The *value* (of length N bits) can be used to directly index the table of 2^N entries.

The required behaviour uses these addressing modes as follows:

- A *key* is presented to the table in order to lookup a *value* + *additional data*. An exact match is required.

- An unrelated *value* is attached to each *key* used for the lookup. If the lookup is successful then the attached *value* is returned with the results.

- If the lookup fails then a new entry is created in the table. The *key* and a default value for the *additional data* are inserted into the table at the entry indexed by the attached *value*. The lookup returns data from this new table entry and a NULL value in place of the attached value.

- The table must be subsequently accessed (indexed by either *key* or *value*) so that the *additional data* field can be fully populated. Until this happens, safe default values for the various QoS parameters are placed in the *additional data* field.

Entry creation which overwrites an existing entry is a required behaviour of the overlapped virtual queueing system. Overwriting is the mechanism of simultaneous queue de-assignment and re-assignment.

A pool of the identities (*value*) of queues which are pending de-assignment is maintained locally by the queue lookup block. As described, values are taken from this pool to append to each lookup. Unused values are returned to pool when the result is returned.

<u>State management</u> :

In order to support the queue lookup block, an additional function is required which can monitor the activity of queues, deassign idle queues, and pass their identities to the queue lookups pool.

This function is most obviously associated with the dequeueing processor (or control logic) as it is during dequeueing that a newly empty queue can be detected. The method of idle queue detection is not the primary focus of this patent therefore it is not described further here. A solution to queue monitoring and deassignment is described in the embodiment section next.

What matters is how the overlap between old and new connections is managed. A key control flag is carried with each packet, referred to as Connection_status. Between queue identification and enqueueing, this flag identifies the first packet of any new connection. Between enqueueing and dequeueing this flag identifies packets belonging to a queue which is pending de-assignment. This information is essential in managing the smooth handover of a queue in a robust manner. For instance, Connection_status could be used in the following ways:

Confidentiality Level: RED Company Confidential

1. Differentiating between the first packet of a new connection and residual packets of an old connection which are buffered between the queue lookup and enqueueing logic at the time of reassignment. When the queue monitoring function re-categroises an idle queue as 'pending deassignment' it must mark that queues state. Some time later, the first packet of a new connection will arrive with the Connection_status flag set. This provides two clear reference points for the queue monitoring an enqueueing logic to work from. In the limbo period in-between, the any function acting on the queue state functions can recognise and conditionally handle any packet belonging to the old connection. For instance, the queue monitoring function may ignore queues which are pending reassignment. The enqueueing function may clear the 'pending' status of the state only when it sees the Connection_status bit set. The status is unaffected by residual packets of the old connection.

2. Differentiating between packets of the new connection and residual packets of the old connection which are backlogged in the queue structure. It is possible that residual packets which are backlogged in the queue memory later upset the Queue State when they are scheduled and the dequeueing function sends a state update. Again, the queue state update function may be configured to deal subtly different with old and new packets.

Finally, note that if a 'long term idle' stream does spring back into life in this way after its queue has been deassigned then the moment the queue is reassigned to a different stream, then the old stream is simply assigned to a new virtual queue. There is thus almost no overhead to virtual queue management and queue switching.

Details of the embodiment

The implementation of overlapped virtual queueing is described in detail in reference [1].

For the queue assignment:

- Content Addressable memory is an ideal memory technology for the table memory.
- The memory is supported by a function which provides and recycles the 'free queue' identities.

For the state management:

- A state engine is an ideal basis for the queue monitoring function. A background system function can be defined which scans through all queue state in the state elements, looking for idle queues. The 'find free queue' algorithm is described further in the state engine patent (?)

Reference material

[1] A. Spencer "Traffic Handler architecture document - V0.1"

- Architecture proposal for ClearSpeed 40G Traffic Handler (in progress)

7.6 Key features of the invention

- After initial assignment, a queue will always belongs to a connection. It either has the state of being either asssigned or pending re-assignment.
- In the pending reassignment state it may still be used by the old connection. Use by newly arriving packets beloinging to the old connection is terminated at the moment of reassignmnet.
- Deassignmnet is implicit in reassignment - there is no explicit messaging involved.
- After reassignment, instead of purging the residual packets of the old connection, those packets are effectively inserted at the start of the new queue. The handover is managed in a controlled fashion to minimise the disruption to the new connection.
- Queues are only deassigned when they are both empty AND there is a demand for resource. There is not measured timeout period.

7.7 Scope of the claim

ClearSpeed

This proposal is only relevant to Traffic Handling application in which many (hundreds of thousands of) queues are required. Specifically, this therefore relates to per-flow Traffic Handling.

Confidentiality Level: RED

Company Confidential

PCT Application
GB0304893

Confidentiality Level: RED
Document Number VRev